Grundlagen der Informatik III

Wintersemester 2013/2014

Prof. Dr.-Ing. Michael Goesele,
Simon Fuhrmann, Fabian Langguth

TECHNISCHE UNIVERSITÄT DARMSTADT

5. Aufgabenblatt

14.11.2013

Die Aufgaben der Präsenzübung sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind **vom 21.11. bis 27.11.** bei Ihrem jeweiligen Tutor zu Ihrer Präsenzübungszeit in handschriftlicher Form abzugeben. Die Abgaben sollen alle zum Nachvollziehen der Lösungen notwendigen Zwischenschritte, sowie den Namen des Tutors und die Übungsgruppe enthalten. Die Abgaben erfolgen in Teams mit drei Personen, wobei alle Mitglieder des Teams derselben Übungsgruppe angehören müssen. Jedes der Mitglieder muss eine eigene, handschriftliche Lösung abgeben und die anderen Mitglieder sind auf den Lösungen deutlich anzugeben.

Aufgabe 1: Arrays, Pointer und Pointerarithmetik

(a) Was macht folgender Code?

```
int *a = new int[1000];
```

(b) Was enthält x nach der Ausführung des folgenden Code-Fragments?

i) Fragment A

```
int* x;
int y = 5;
x = &y;
```

ii) Fragment B

```
int x = 3;
int* y;
y = &x;
*y = 42;
```

(c) Geben Sie C oder C++ Code an, der ein Array von Integern der Länge vier erstellt und dieses mit den Werten 1-4 füllt. Geben Sie dabei drei unterschiedliche Varianten an, um die Initialisierung vorzunehmen: Mit Initialisierungslisten, mit dem []-Operator und sowie mit Pointerarithmetik.

Aufgabe 2: Templates

(a) Verallgemeinern Sie folgende Funktion mit Hilfe von Templates so, dass sie das Maximum zweier Instanzen eines beliebigen Datentyps zurückgibt. Sie dürfen den >-Operator verwenden. D.h. Sie können davon ausgehen, dass der von Ihnen verwendete Typ entweder primitiv ist, oder den >-Operator überlädt.

(b) Verallgemeinern Sie folgende Funktion mit Hilfe von Templates so, dass sie zwei Instanzen eines beliebigen Datentyps vertauscht.

```
void swap(int& a, int& b);      // vertauscht die beiden Zahlen a und b
```

(c) Gegeben ist folgender C++ Code:

```cpp
#include <iostream>
using namespace std;

template<typename T, int n> T sum(T* arr){ // summiert alle Elemente von arr auf
    T result(0);
    for(int i=0; i<n; i++)
        result += arr[i];
    return result;
}
int main() {
    const int length = 22;
    int a[length];                                  // Array erzeugen
    for(int i=0; i<length; i++)                     // initialisieren
        a[i]=i;
        cout << "␣sum:␣" << sum<int,length>(a) << endl; // Ausgabe
}
```

Welche Funktion erfüllt der Template-Parameter n?
Was ist der Vorteil daran ihn als Template- statt als normalen Parameter zu übergeben? Was ist der Nachteil?

Aufgabe 3: Vererbung

Gegeben sei folgender C++ Code, der eine Klassenhierarchie definiert.

```cpp
class A
{
public:
    ~A() { std::cout << "Destruktor␣A\n" ; }
    virtual void f() = 0;
    void g() { std::cout << "A::g()\n" ; }
};
class B : public A
{
public:
    B() { }
    ~B() { std::cout << "Destruktor␣B\n" ; }
    virtual void f() { std::cout << "B::f()\n" ; }
    void g() { std::cout << "B::g()\n" ; }
};
class C : protected virtual B
{
public:
    void h() { std::cout << "C::h()\n" ; }
};
```

(a) Können Sie eine Instanz vom Typ A anlegen? Begründen Sie.

(b) Welche Ausgabe erzeugt der folgende Code

```cpp
B b;
b.f();
b.g();
```

(c) Welche Ausgabe erzeugt der folgende Code

```cpp
A* b = new B;
b->f();
b->g();
```

```
delete b;
```

(d) Wie können Sie die Deklarationen von A bzw. B ändern, damit der Code in Teilaufgabe (b) und (c) dieselbe Ausgabe erzeugt? Diskutieren Sie die Vorteile der neuen Deklaration für den Fall von dynamischer Speicherallozierung innerhalb von B.

(e) Ist folgende Zuweisung zulässig? Können von der Klasse C überhaupt Instanzen erzeugt werden?

```
A* c = new C;
```

(f) Welche der Methoden f(), g() und h() der Klasse C sind von außen sichtbar?

Aufgabe 4: Template-Klasse Vector

Schreiben Sie eine Template-Klasse Vector, die einen Vektor kapseln soll. Die Klasse soll sowohl über den Elementtyp (z.B. double, int, ...) als auch über die Dimension (z.B. 2, 3, ...) getemplated sein und im Header `vector.h` definiert sein. Die Klasse soll

1. einen Konstruktor Vector() besitzen, der den Inhalt mit Nullen initialisiert.
2. den Index-Operator [] überladen, damit man auf die einzelnen Elemente des Vektors zugreifen kann. (Hinweis: Eine Funktion kann Referenzen zurückgeben)
3. den Additionsoperator + überladen, so dass man mit den entsprechenden Aufrufen Vektoren addieren kann.
4. den Multiplikationsoperator * sowohl für skalare Elemente als auch für Vektoren überladen. Im ersten Fall soll eine Skalarmultiplikation durchgeführt, im zweiten Fall ein Skalarprodukt berechnet werden.
5. eine Funktion length() besitzen, die einem die Länge des Vektors (euklidische Norm) zurückgibt.
6. eine Funktion print() besitzen, die den Inhalt des Vektors auf dem Bildschirm ausgibt.

Verwenden Sie zum Testen das unten angegebene Codegerüst.

```
#include <iostream>
#include "vector.h"

int main()
{
    typedef Vector<float, 3> Vector;
    Vector f, g;
    f[0] = 1.0f; f[1] = 2.0f; f[2] = 3.0f;
    g[0] = 3.0f; g[1] = 2.0f; g[2] = 1.0f;

    Vector h = f + g;
    Vector m= g * 3.0f;

    f.print(); // 1 2 3
    g.print(); // 3 2 1
    h.print(); // 4 4 4
    m.print(); // 9 6 3
    std::cout << (f * g) << '\n'                // 10
              << f.length() << std::endl;       // 3.74166

    return 0;
}
```

Hausaufgabe 1: Vererbung - Rock, Paper, Scissors (3 Punkte)

Weiter unten finden Sie ein Codegerüßt, dass die Klasse `Tool` zur verfügung stellt und testet. Erweitern Sie diese Klasse um einen Integer `strength` und einen Char `type`. Die Klasse soll außerdem die Funktion `setStrength(int)` implementieren, welche die Variable `strength` setzt.
Implementieren Sie weiterhin die 3 Klassen `Rock`, `Paper` und `Scissors`, die von der Klasse `Tool` erben. Jede dieser Klassen soll einen Konstruktor besitzen, der einen Integer übergeben bekommt und damit das entsprechende `strength` Feld der Basisklasse initialisiert. Außerdem soll jeder Konstruktor den Char `type` entsprechend setzen.
Die Klassen benötigen weiterhin eine public Funktion `bool fight(Tool)` um gegen Gegner antreten zu können. Die Funktion soll `true` zurückgeben falls die aufrufende Klasse gewinnt, und `false` falls sie verliert. Um die Spielregeln von Rock, Paper, Scissors nachzubilden sollen Sie folgende Eigenschaften in den Funktionen der einzelnen Klassen implementieren:

- Die `strength` von Rock wird temporär verdoppelt, wenn Rock gegen Scissors kämpft.
- Die `strength` von Rock wird temporär halbiert, wenn Rock gegen Paper kämpft.
- Die `strength` von Paper wird temporär verdoppelt, wenn Paper gegen Rock kämpft.
- Die `strength` von Paper wird temporär halbiert, wenn Paper gegen Scissors kämpft.
- Die `strength` von Scissors wird temporär verdoppelt, wenn Scissors gegen Paper kämpft.
- Die `strength` von Scissors wird temporär halbiert, wenn Scissors gegen Rock kämpft.

Sollten Sie Gefallen an der Aufgabe finden können Sie das Programm zusätzlich um die Klassen `Lizard` und `Spock` erweitern (hierfür gibt es keine zusätzlichen Punkte) `:-)`

```
#include <iostream>

class Tool {
    /* Fill in */
};

/*
    Implement class Scissors
*/

/*
    Implement class Paper
*/

/*
    Implement class Rock
*/

int main(void)
{
    // Example main function

    Scissors s1(5);
    Paper p1(7);
    Rock r1(15);
    cout << s1.fight(p1) << p1.fight(s1) << endl;
    cout << p1.fight(r1) << r1.fight(p1) << endl;
    cout << r1.fight(s1) << s1.fight(r1) << endl;

    return 0;
}
```

Hausaufgabe 2: Template-Klasse Matrix (5 Punkte)

Schreiben Sie eine Template-Klasse `Matrix`, die eine Matrix kapseln soll. Die Klasse soll sowohl über den Elementtyp (z.B. double, int, ...) als auch über die Dimensionen (z.B. 2, 3, ...) getemplated sein. Eine 2x3 Matrix (2 Reihen, 3 Spalten) mit dem Datentyp `float` soll folgendermaßen deklariert werden können: `Matrix<float,2,3>` Die Klasse soll

1. einen Konstruktor `Matrix()` besitzen, der den Inhalt mit Nullen initialisiert. (0.5 Pkt)
2. den Operator `()` überladen, damit man auf die einzelnen Elemente der Matrix in der Form (Reihe, Spalte) zugreifen kann. (Hinweis: Eine Funktion kann Referenzen zurückgeben) (0.5 Pkt)
3. den Additionsoperator + überladen, so dass man mit den entsprechenden Aufrufen Matrizen addieren kann. (0.5 Pkt)
4. den Multiplikationsoperator * für Matrizen überladen sodas er das Produkt zweier zueinander passender Matrizen berechnet. Achten Sie hierbei auf die jeweiligen Größen der Eingabe- und Ausgabematrizen. (1 Pkt)
5. eine Funktion `min()` besitzen, die das Minimum der Matrix zurückgibt. (0.5 Pkt)
6. eine Funktion `max()` besitzen, die das Maximum der Matrix zurückgibt. (0.5 Pkt)
7. eine Funktion `print()` besitzen, die den Inhalt der Matrix auf dem Bildschirm ausgibt. (0.5 Pkt)

Implementieren Sie außerdem auch eine spezialisierte Funktion `determinant(Matrix)` NUR für 3x3 Matrizen, welche die Determinante einer übergebenen 3x3 Matrix berechnet. (1 Pkt)
Stellen Sie ihren Code in einer Datei `matrix.h` zur Verfügung und verwenden Sie zum Testen das unten angegebene Codegerüst:

```
#include <iostream>

#include "matrix.h"

int main(void)
{
    Matrix<int, 3, 3> A;
    A.print();
    std::cout << std::endl;

    A(0,0) = 1; A(0,1) = 2; A(0,2) = 3;
    A(1,0) = 1; A(1,1) = 2; A(1,2) = 3;
    A(2,0) = 1; A(2,1) = 2; A(2,2) = 3;

    Matrix<int, 3, 3> B;
    B(0,0) = 1; B(0,1) = 2; B(0,2) = 3;
    B(1,0) = 4; B(1,1) = 5; B(1,2) = 6;
    B(2,0) = 7; B(2,1) = 8; B(2,2) = 9;

    A.print();
    std::cout << std::endl;
    B.print();
    std::cout << std::endl;

    Matrix<int, 3, 3> C = A + B;

    C.print();
    std::cout << std::endl;


    Matrix<int, 4, 3> D, E;
    D(0,0) = 1; D(0,1) = 2; D(0,2) = 3;
    D(1,0) = 4; D(1,1) = 5; D(1,2) = 6;
```

```
    D(2,0) = 7; D(2,1) = 8; D(2,2) = 9;
    D(3,0) = 7; D(3,1) = 8; D(3,2) = 9;

    D.print();
    std::cout << std::endl;

    E = D * A;
    E.print();
    std::cout << std::endl;

    std::cout << E.min() << "␣" << E.max() << std::endl; // Ausgabe: 6 72

    Matrix<int, 3, 3> F;
    F(0,0) = 2; F(0,1) = 1; F(0,2) = 1;
    F(1,0) = 1; F(1,1) = 2; F(1,2) = 1;
    F(2,0) = 1; F(2,1) = 1; F(2,2) = 2;

    std::cout << determinant(F) << std::endl; // Ausgabe 4

    return 0;
}
```

Hausaufgabe 3: STL - Template-Funktion toStr() (2 Punkte)

Implementieren Sie die Funktion `toStr`. Die Funktionssignatur soll folgende sein:

```
template<typename T> std::string toStr(const T & value);
```

Die Funktion soll `std::stringsteam` benuzten um aus einem beliebigen Wert einen String zu generieren. Ziel dieser Übung ist es, dass sie sich mit der C++ STL Refrenz vertraut machen - Sie finden alle benötigten Informationen zu `std::stringstream` unter `http://www.cplusplus.com/reference/sstream/stringstream/`.
Was ist der Vorteil dieser Template Funktion gegenüber einzelnen, überladenen Funktionen für jeden Datentyp `T`?